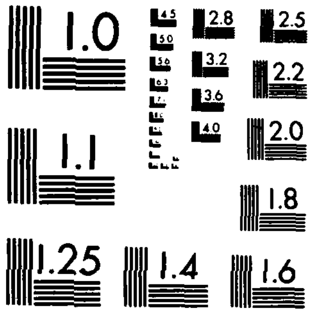
AD-A141 063 COBOL AUTOMATED VERIFICATION SYSTEM IMPLEMENTATION PHASE(U) GENERAL RESEARCH CORP SANTA BARBARA CA R A MELTON ET AL. DEC 83 GRC-CR-10-970 RADC-TR-83-284 1/1

UNCLASSIFIED F30602-80-C-0101 F/G 9/2 NL

END
DATE
FILMED
6-84
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

RADC-TR-83-284
Final Technical Report
December 1983

COBOL AUTOMATED VERIFICATION SYSTEM, Implementation Phase

General Research Corporation

Richard A. Melton and William R. Wisehart

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

APPROVED: [illegible]

LAWRENCE M. LOMBARDO
Project Engineer

APPROVED: [illegible]

RAYMOND P. URTZ, JR.
Acting Technical Director
Command and Control Division

FOR THE COMMANDER: [illegible]

JOHN A. RITZ
Acting Chief, Plans Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-83-284	2. GOVT ACCESSION NO. AD-A141063	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) COBOL AUTOMATED VERIFICATION SYSTEM Implementation Phase		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report October 1981 - May 1983
		6. PERFORMING ORG. REPORT NUMBER CR-10-970
7. AUTHOR(s) Richard A. Melton William R. Wisehart		8. CONTRACT OR GRANT NUMBER(s) F30602-80-C-0101
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Research Corporation P.O. Box 6770 Santa Barbara CA 93111		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63728F 25270208
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (COEE) Griffiss AFB NY 13441		12. REPORT DATE December 1983
		13. NUMBER OF PAGES 48
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Lawrence M. Lombardo (COEE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

COBOL
Software Quality
Software Testing
Interactive
Automated Documentation
Static Analysis
Dynamic Analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

CAVS (for COBOL Automated Verification System) is a menu-driven, integrated, and interactive set of tools to improve the productivity of COBOL programmers.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

CONTENTS

1 TECHNICAL PROBLEM

As the cost of computer hardware continues to fall, more and more ambitious software applications are conceived. The conversion of the ambitious conceptions into complete and reliable final products, on budget and within schedule, continues to be an elusive goal for the majority of projects. Much emphasis is being placed on solving this problem with tools and techniques that are applicable throughout the software life cycle. Following is a partial list of the approaches currently being actively pursued.

1. Automated tools for program analysis and testing - this is the approach which best describes CAVS.

2. Fourth generation languages - usually described as non-procedural languages. The biggest impact so far has been in data base applications (as a more productive replacement for COBOL).

3. Software management and configuration control tools - a must for large software developments.

4. Software support systems/programming environment tools - easy to use, integrated environments. The best known examples are UNIX, INTERLISP, SMALLTALK.

5. Tools for writing and analyzing requirements/design specifications - the earlier an error is detected the less it costs to fix it.

6. Programming workstations - the next step after timesharing - eliminating bottlenecks due to saturation of timesharing systems. The falling cost and rising performance of microcomputer systems is making it economically feasible to provide software developers with dedicated workstations which can be networked together for large software development projects.

7. Software engineering life cycle support systems - putting all the pieces together in an easy to use, integrated framework for all phases and all personnel during the entire software life cycle.

This project falls in the first area, automated tools for program analysis and testing. The objective was to design and implement a tool for analyzing and testing COBOL source programs. The tool, called CAVS, for COBOL Automated Verification System, is a prototype of a software tool to improve the reliability and maintainability of COBOL software systems. CAVS can be applied during the testing, verification, validation, and error detection/correction phases of software development.

2 GENERAL METHODOLOGY

This effort was accomplished in two phases. Phase I consisted of a study of the COBOL programming language and of the latest automated software test and verification tools, to insure that the most advanced techniques and capabilities would be included in this development. The output of Phase I was a CAVS Initial Design. A Functional Description,[1] System/Subsystem Specification,[2] and Final Report[3] were produced during Phase I.

Phase II consisted of the detailed design, implementation, and testing of CAVS on the VAX 11/780/VMS computer system at General Research Corporation, Santa Barbara, California. CAVS was then installed on the VAX 11/780/VMS and Honeywell H6180/GCOS computer systems at the Rome Air Development Center (RADC), Griffiss AFB, N.Y. It was also installed on the UNIVAC 1100/EXEC 8 computer systems at the Defense Mapping Agency (DMA) Hydrographic/Topographic Center (DMAHTC) in Washington, D.C., at the DMA Aerospace Center (DMAAC) in St. Louis, Mo., and at the Navy Data Automation Facility (NAVDAF) in Newport, Rhode Island.

After installations were completed, user training courses were conducted at the two DMA sites. A maintenance training course was conducted at DMAHTC. Two important documents from Phase II were the User's Manual[4] and the Program Maintenance Manual.[5]

---

[1] M. Sharp, et al., COBOL Automated Verification System Functional Description, General Research Corporation CR-2-970, November 1980.

[2] R. Melton, et al., COBOL Automated Verification System:System/Sub-System Specification, General Research Corporation CR-1-970, November 1980.

[3] G. Greenberg, et al., COBOL Automated Verification System Final Report: Study Phase, General Research Corporation CR-3-970, November 1980.

[4] R. Melton, et al., COBOL Automated Verification System User's Manual, General Research Corporation CR-4-970/R1, August 1983.

[5] R. Melton, et al., COBOL Automated Verification System Program Maintenance Manual, General Research Corporation, CR-7-970/R1, July 1983.

3 TECHNICAL RESULTS

A prototype AVS for COBOL was produced as a result of the project. CAVS is an integrated set of tools that builds a database from COBOL source, and then accesses the database for documenting, analyzing, and testing COBOL programs. CAVS processes six dialects of COBOL: ANSI Standard 1968 and 1974, UNIVAC FIELDATA, UNIVAC ASCII, Honeywell 60/6000, and Honeywell COBOL-74. CAVS is currently installed on a VAX 11/780 under VMS, on a UNIVAC 1100/60 under EXEC 8, and on the Honeywell 6180 under GCOS.

The principal functions performed by CAVS are:

- Static Analysis, in which multi-module checking is performed for errors and dangerous coding practices not normally detected by COBOL compilers.
- Instrumentation, which consists of automatic insertion into a COBOL program of calls to data collection routines that record data about program operations and paths taken during execution.
- Trace File Analysis, which combines the program information from the CAVS database with the trace file information created by running an instrumented program to produce performance reports about the program's dynamic operation.
- Documentation, which generates a number of reports from the CAVS database about a program's structure and usage of variables. These reports are of use to developers, testers, and maintainers.
- Precompilation of a COBOL language extension that includes "end verb" statements (similar to the suggested COBOL 8x syntax) into standard COBOL.
- Reformatting of COBOL source code with indentation of nested IF verbs and standardized spacing of the clauses of other verbs for improved readability.

Features of CAVS include:

- Interactive operation with menus, help screens, menu commands, wildcards in name selection, filename prompts, selective reports
- Batch operation with commands for processing large amounts of source code, and generating comprehensive reports
- Automatically updated library of COBOL source code, reports about it, and testing history
- Efficiency to support frequent use on medium to large COBOL systems
- Applicable to systems of up to 250,000 lines of COBOL source

Interactive Mode

CAVS provides menus that tell the user which functions and reports are available. For novice users, one or more help screens a.e available from each CAVS menu. Experien^ed CAVS users can suppress the display of CAVS menus by entering sequences of menu selections. Frequently used sequences of menu selections can be named and saved as commands. Wildcards provide a convenient way of selecting specific names (program-IDs, data items, literals, paragraphs, files/records, copy members/records, and global records), as well as finding all names available. Filename prompts provide flexibility in reading in COBOL source elements, generating concise reports, and outputting COBOL source elements. Viewing reports interactively before sending them to a report file allows very selective reports to be generated.

Figure 3.1 gives a broad overview of the CAVS functions in terms of the menus that the user interacts with to control the functions. The chart is arranged in hierarchical order by the connectivity from one menu to another. The WELCOME menu controls initialization, the FUNCTIONS menu controls access to the main function menus, and the END OF CAVS controls the session wrapup.

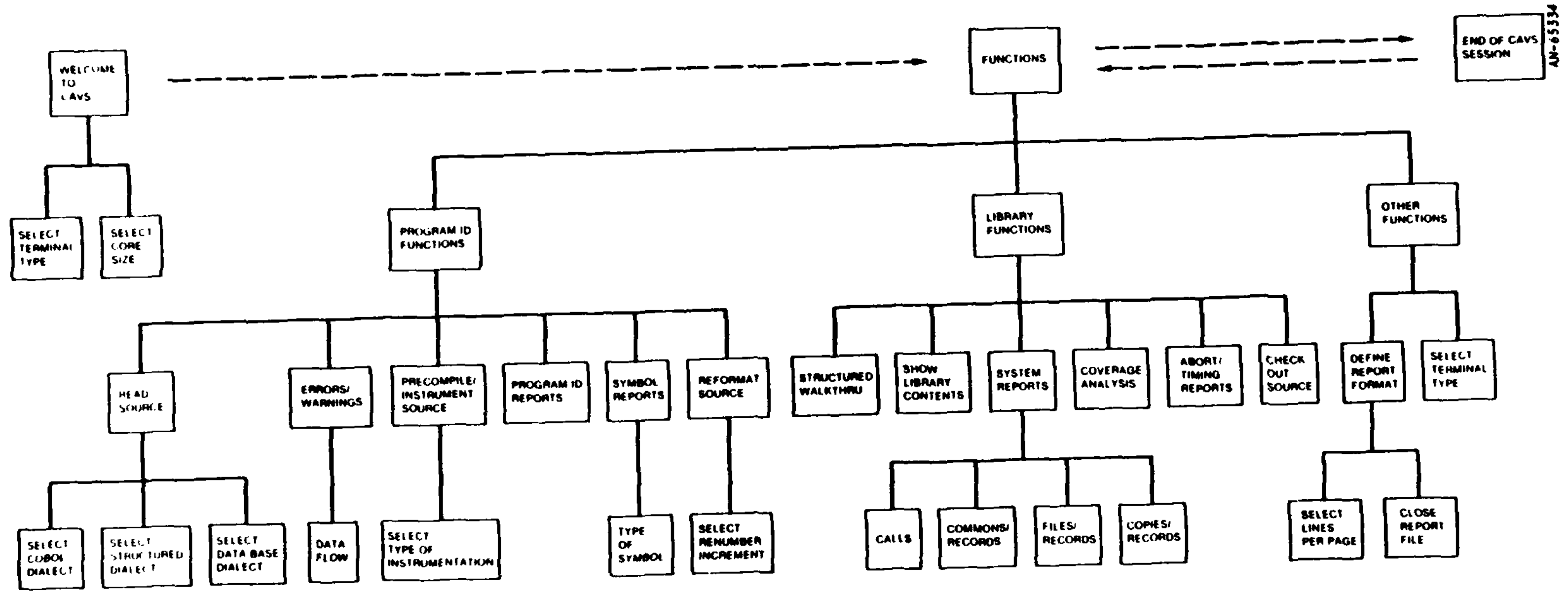


Figure 3.1. CAVS Menu Hierarchy

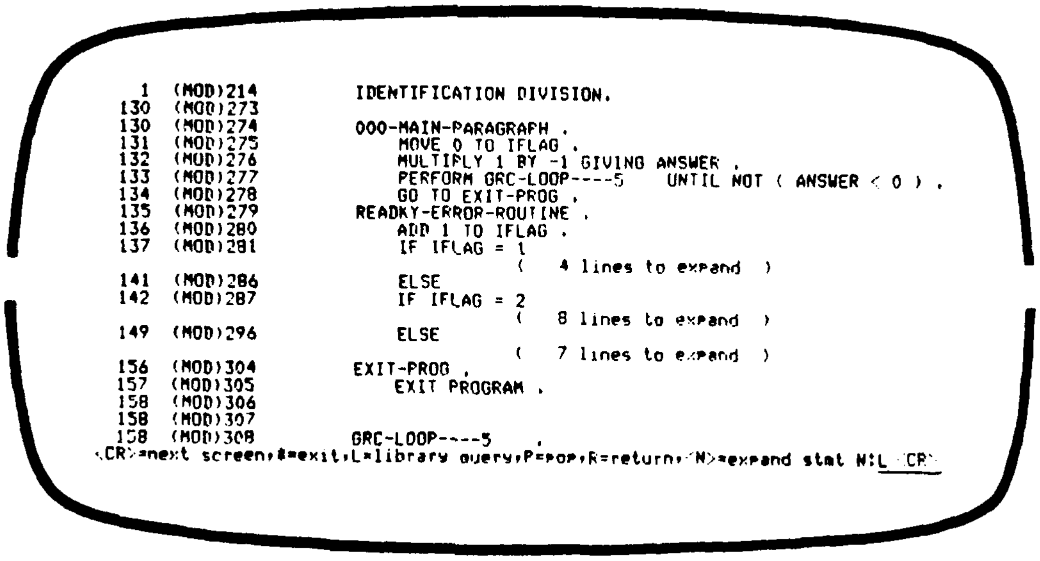
Batch Mode

Batch mode is provided for processing large source elements, or generating comprehensive reports for baseline, archival documentation. Batch commands make it easy to direct CAVS in batch mode because they relate directly to the CAVS output desired rather than the sequence of processing CAVS goes through to produce the output. The CAVS batch commands are listed below:

```
[PROJ[ECT LIBRARY].]
[EXPA[ND].]
[CHAN[GE].]

{OPTI[ON] = option {,option}.}
          where option is one of

     ABOR[T]             LIST
     CHEC[K OUT]         PREC[OMPILE]
     DOCU[MENT]          REAC[HING SET]
     ERRO[RS]            REFO[RMAT]
     INST[RUMENT]        SUMM[ARY]
     IOCO[UNTS]          TIMI[NG]

{REPO[RT] = report {,report}.}
     where report is one of:

     CALL[S]                            SUMMARY
     GLOB[AL VARIABLES]                 BRANCHES
     PROF[ILE]                          STATEMENTS
     GLOB[AL VARIABLES]/E[NHANCED]      TRACE
     IO                                 TIME
     PROP[ERTIES]
     CROS[S]
     CALL[S]/T[EXT]
     BAND[S/<n>]
     PICT[URE]

{FOR P[ROGRAM-IDS] = <name1><name2>...
{TEST[CASE] = <name>
{REAC[HING SET],PROG[RAM-ID = (<name>),
     TO = <statement number>,
     FROM = <statement number>,}

     [ ] optional
     { } optional an arbitrary number of times
     < > integer constant or character string
```

Support for Structured Programming in COBOL

CAVS supports structured programming in COBOL in a number of ways. COBOL constructs (data division and procedure division) are automatically indented on all hard copy source listings. Source listings also include a perform range cross reference at the end of each PROGRAM-ID. CAVS also provides the option to automatically reformat COBOL source text to make it easier to read (including indentation of procedure division constructs). Structured COBOL written in an extension to COBOL is recognized by CAVS and can be automatically precompiled into standard COBOL. The major support for structured programming in COBOL is the CAVS structured walkthrough. This is an innovation in CAVS which allows you to follow the source code of very large programs at a video terminal, using the program structure as the guide to examining the code. Top levels of the structure are displayed first, and with simple commands you can expand to successively lower levels, and then easily return to where you started.



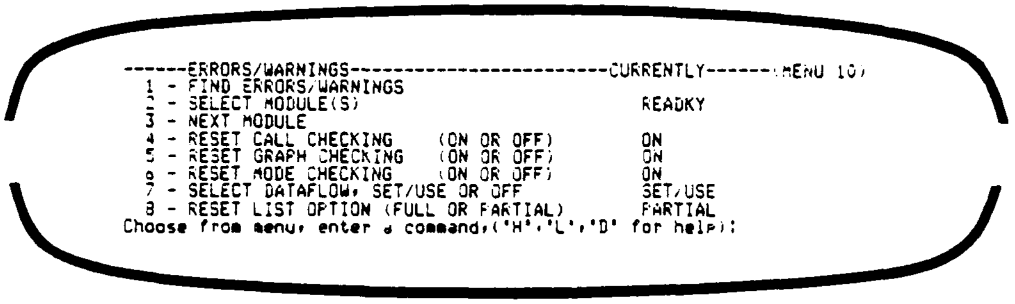
```
  1  (MOD)214          IDENTIFICATION DIVISION.
130  (MOD)273
130  (MOD)274          000-MAIN-PARAGRAPH .
131  (MOD)275              MOVE 0 TO IFLAG .
132  (MOD)276              MULTIPLY 1 BY -1 GIVING ANSWER .
133  (MOD)277              PERFORM GRC-LOOP----5    UNTIL NOT ( ANSWER < 0 ) .
134  (MOD)278              GO TO EXIT-PROG .
135  (MOD)279          READKY-ERROR-ROUTINE .
136  (MOD)280              ADD 1 TO IFLAG .
137  (MOD)281              IF IFLAG = 1
                                   (    4 lines to expand  )
141  (MOD)286              ELSE
142  (MOD)287              IF IFLAG = 2
                                   (    8 lines to expand  )
149  (MOD)296              ELSE
                                   (    7 lines to expand  )
156  (MOD)304          EXIT-PROG .
157  (MOD)305              EXIT PROGRAM .
158  (MOD)306
158  (MOD)307
158  (MOD)308          GRC-LOOP----5    .
<CR>=next screen,#=exit,L=library query,P=pop,R=return,<N>=expand stat N:L <CR>
```

Error/Warning Reports (Static Analysis)

CAVS provides the following static analysis functions:

- Call checking to detect errors in the number of calling parameters or picture mismatch between parameters.
- Symbol set/use checking to detect data items which are used but not set or set but not used.
- Data flow analysis to detect data items which are used before being set on all paths, reset before being used, or set and then not used on a particular path.
- Structure checking to detect unreachable code and potential infinite loops.
- Picture checking to detect implicit picture conversions in move statements.

Each of these checks may be turned on or off from the menu.



```
------ERRORS/WARNINGS------------------------CURRENTLY------(MENU 10)
  1 - FIND ERRORS/WARNINGS
  2 - SELECT MODULE(S)                           READKY
  3 - NEXT MODULE
  4 - RESET CALL CHECKING     (ON OR OFF)        ON
  5 - RESET GRAPH CHECKING    (ON OR OFF)        ON
  6 - RESET MODE CHECKING     (ON OR OFF)        ON
  7 - SELECT DATAFLOW, SET/USE OR OFF            SET/USE
  8 - RESET LIST OPTION (FULL OR PARTIAL)        PARTIAL
Choose from menu, enter a command,('H','L','D' for help):
```

Dynamic Analysis

Dynamic analysis is a three step process:

- Automatic COBOL source code instrumentation using CAVS
- Compilation/execution/trace file generation
- Trace file analysis using CAVS

CAVS performs abort tracing instrumentatior timing instrumentation, I/O instrumentation, and statement/branch instrumentation. After executing the instrumented COBOL and generating a trace file, CAVS can be used to produce abort trace reports, timing reports, or statement/branch coverage reports. Statement/branch coverage results can be saved on the project library and accumulated to provide a testing history.

Documentation

CAVS provides a wide variety of documentation reports in both batch and interactive mode. They include detailed reports about PROGRAM-IDs, symbol properties, symbol cross references, and source text reports. System-wide reports are provided on PROGRAM-IDs, ENTRY POINTs, global records (in COMMONs, FILEs, or COPYs), and TESTCASES.

Automatically Updated Library

CAVS automatically updates the project library so that it is as current as the last source code processed. A baseline library can be used for baseline source code, reports, and testing history. Private libraries can be created and maintained economically, saving only changes to the baseline library on change files.

Efficiency -- What Does it Cost?

CAVS processes source code (read and create/update a project library) is about 1.5 times compile time. The CAVS project library is comparable to source code size (about the same size as your source code stored in 80 column card images). CAVS reports are easy to use, and are often better than compiler listings or text editor searches for solving specific problems. Selective reports generated in menu mode are especially cost-effective. The cost of CAVS dynamic analysis is highly dependent on the logic of the program. In general it causes about 20% code expansion and 50% CPU overhead.

Applicable to Large Software Systems

CAVS is especially cost-effective for large software systems consisting of many interrelated programs or subroutines. It is usually too expensive to reanalyze a whole system in order to reanalyze one or two modified subroutines. The CAVS project library contains inter-relationships between all programs, subroutines, files, and copy members. Modified subroutines can be reanalyzed separately, and the entire library automatically updated at the same time.

4 IMPLICATIONS FOR FURTHER RESEARCH

The continuing research in source code analysis systems sponsored by RADC demonstrates that the concepts are largely independent of the language the source code is written in. The following comments are derived from experience with CAVS, but the implications of that experience for future research apply to any language.

Experience has shown that the AVS features embodied in CAVS are useful, and in particular, that interactive operation combined with user selection of report scope leads to more efficient use of an AVS. As a result of using the system, the following statements can be made that point the way to future research.

- There is a wealth of information in an AVS data base, and more could be added. Many useful reports could be generated that are not now available from existing or new information in the data base.
- Interactive users would benefit from more modes of viewing the reports. Spreadsheet calculators have shown the way to concepts of moving one or more "windows" over a large report to view or alter the contents.
- Even more sophisticated viewing techniques could be applied to some reports as has been done in the structured walkthrough. In structured walkthrough the user can follow complex source code, guided by the code structure. CAVS knows the meaning of the structure and makes it easy to walk through it. Other reports could use content to guide the user, and there are several other "maps" that could be used to guide the source code walkthrough.
- Demonstration of the utility of CAVS as an interactive tool leads naturally to the idea of distributing the tool in a "workstation" environment. One central data base would be maintained, and users at work stations could retrieve sections of it to their own work station for viewing. Changes could be made to the central data base as well, but security measures would have to be introduced.

- The ultimate goal of all this research should be the development of a "life cycle" environment in which the data base starts with the requirements definition and builds from there. Many tools would have access to the data base, which includes all the history of the evolution of the code.

These broad areas are expanded in the following sections.

Section	Topic
4.1	New Reports and Analyses
4.2	Enhanced User Interface
4.3	Expanded Walkthrough Capability
4.4	AVS Tool Technology in a Life Cycle Environment

4.1 NEW REPORTS AND ANALYSES

4.1.1 Enhancements to Matrix Reports

The matrix report summarizes the usage of global variables (rows) across several modules (columns). A possible enhancement would be to use the matrix display for interactively selecting a global variable and a program-ID for more detailed display. The user could select a row/column element, a whole row (one variable in all modules), or a whole column (all global variables in one module). Additional information that could be displayed includes statement cross reference numbers, statement cross reference text, context report, and a properties report. CAVS users at low-speed video terminals or hardcopy terminals would get large matrices split across several screens as they currently do. An interactive user at a high-speed video terminal could scroll horizontally (across modules) or vertically (across global variables) to view a matrix larger than one screen.

Note that this approach can be applied to a variety of matrix reports - symbol used/symbol set, calling module/called module, testcase/module, copy text/module, file/module. The wildcard idea could be used to build the matrix according to the user's needs (row names of interest/column names of interest) thus providing a natural way to limit the amount of information presented.

4.1.2 Dynamic Analysis Enhancements

Statement coverage and hit/not-hit reports could be done in the same fashion as the structured walkthrough. For a coverage report, the statement execution counts would appear on the lefthand side. For hit/not hit reports, a yes/no indicator would appear on the lefthand side. With suitable video terminals, statements not executed could be displayed in brighter intensity or reverse video, or even in a different color (red for instance).

While a trace file is being read, video terminal users could be presented with a display of the dynamic calling tree, down to the perform range/block level. This applies to both abort trace files and coverage trace files. The source code of the program that was executed as the trace file was created could also be displayed (see the discussion of TRACE WALKTHROUGH/WALKBACK in Sec. 4.3.2).

A method of coverage instrumentation/analysis which reduces the size of CAVS trace files by a factor of 20 to 100 was designed and partially implemented during Phase II of the CAVS project. This implementation will have to be completed before CAVS can be effectively used in coverage testing of large COBOL systems.

Programs that nearly fill memory may become too large to run when the entire program is instrumented. One approach to this problem is to add a means of merging the trace files produced by the instrumented code. The program could then be run (with the same data), in several versions with different portions instrumented, and then the trace files merged. In this way, comprehensive test reports can be obtained for any size of system. The merge capability would be an enhancement to the current CAVS dynamic analysis capabilities.

A new standard for COBOL (COBOL/80x) has been proposed and is slowly being accepted by the COBOL community. It includes a major extension to the structured control constructs of COBOL. CAVS can contribute to a smooth transition to the new standard by precompiling the new structured constructs into today's standard COBOL. New COBOL code could be written with the new constructs, and old code could be revised to use the new constructs, before upgraded COBOL compilers are available.

A noteworthy trend in newer programming languages is the inclusion of executable assertions. The same thing can be accomplished for existing languages by precompiling assertions into standard code. The CAVS COBOL precompiler could be enhanced to support executable assertions for COBOL. A further enhancement to CAVS would tie assertion violations to the trace file created by an instrumented COBOL program. Dynamic testing reports would then be able to trace the path to an assertion violation (under the kinds of interactive user control described in Sec. 4.3.2 - TRACE WALKTHROUGH/ WALKBACK).

4.1.3 Module Documentation Enhancements

The module documentation report has proven useful in self-documenting CAVS, FAVS, and J73AVS. CAVS now provides this as an on-line report, which is as up-to-date as the project library from which it is generated. But it can be more effectively used in interactive mode if it becomes the top of a set of reports (XREF, TEXT, and CONTEXT) which the user can request as he needs to:

1. For a specified formal parameter.
2. For a specified common variable.
3. For each external called or each calling routine.
4. For a specified file in the routine.

Additional information to include in these reports is:

1. Copy texts included.
2. Block/perform range structure report.

3. Testcases exercising this module.

4.1.4 Library Contents Enhancements

The library contents report should expand into more detailed information at the discretion of the interactive user. For instance the undefined entry points could be defined as stubs by an interactive user. This report also needs to have the date and time when modules were read in; this information could probably also go on the heading of hardcopy source listings.

4.1.5 Automatic Restructuring

GRC's Fortran Automatic Verification System contains a function for automatically restructuring FORTRAN programs to eliminate GOTOs. There are several problems with the restructuring procedure as it now exists:

- Automatic restructuring without additional human input produces an equally bad program without GOTOs.
- Restructuring a program causes documentation of the old program's internal logic to become obsolete.
- Restructuring large programs is time-consuming and expensive.

Drawing on the experience of users at DMA and GRC, we believe that an improved, cost-effective restructuring module can be built, to add COBOL restructuring to CAVS. It should provide these services:

- Automatic analysis of existing program structure.
- Easy-to-read program graph reports. These would show how to break a large program into chunks that are easily understood by people and easily structured by computer.
- Error and diagnostic reports. Any COBOL constructs that render the program unstructurable would be reported.
- Automatic restructuring of COBOL source programs. CAVS would be able to restructure any COBOL dialect it could recognize.

- Automatic chart generation. Documentation (VTOCs and/or flow-charts) for newly structured programs could be generated mechanically. Cost of documenting these new versions of well-tested programs would be reduced.

4.1.6 Automatic Documentation

CAVS is designed to be portable across computer models and operating systems. Its automatic documentation modules concentrate on the portable portions of COBOL programs. But information about nonportable language constructs, file structures, and program interfaces can also be important. During conversion from one computer to another or one operating system to another, knowledge of these non-portable system characteristics is vital. Conversions can be simplified and expedited when this information is extracted and presented automatically.

4.1.7 Management Level Functions

The AVS can be made intelligent enough to:

- Track the progress of modules through the system.
- Report their status to users and management.
- Suggest courses of action.
- Explain what happens when a particular course is selected.
- Search and manipulate CAVS report output in ways tailored to a specific user-defined problem.

All of these functions should be available to on-line users.

HISTORY is a proposed system-wide data logging function. It would record:

- When modules were added to a library.
- How many versions of a module had been added.

- If and when a module was analyzed, documented, instrumented, or tested.
- Which users and projects were actually using CAVS, and to what degree.

Status displays for managers and programmers would reveal the current state of program development for an entire project or for a single program. How the tool was actually used could then be more accurately determined. Th AVS should collect enough data about its own behavior to guide managers and programmers in the most effective use of the tool.

The AVS could use its HISTORY information to SUGGEST courses of action to on-line users. On-line users are already guided through a work cycle. Program development also follows a predictable cycle of coding, analyzing, testing, and documentation. The AVS could look at a module's status and propose what actions should be done next.

SUGGEST and HISTORY would relieve experienced programmers of much of the bookkeeping done during development and maintenance. Inexperienced programmers and new AVS users could become productive more quickly. If the suggestions were accepted, the AVS could begin executing the function immediately or generate a job to execute the function in batch mode.

4.2 ENHANCED USER INTERFACE

Enhancements to the CAVS user interface can be categorized into two classes:

1. Terminal-dependent enhancements - such as horizontal/vertical scrolling, and multiple windows. These techniques are best implemented on high-speed video terminals and could substantially improve the usefulness of CAVS on those terminals.
2. CAVS-dependent enhancements - making all CAVS reports interactive in a sense similar to the structured walkthrough. These techniques would apply to interactive users of CAVS at any kind of terminal.

Note that neither of these approaches excludes the other. They would complement each other for CAVS users at high-speed video terminals.

4.2.1 Scrolling

Interactive reports are generally limited to one screenful at a time - 80 characters by 24 lines. One way to overcome this size limitation is to provide scrolling (horizontal and vertical) within the tool generating the reports. The report format can still be too large to put on the screen all at one time, but the interactive user can 'move' the screen to areas of interest. This approach is limited to cursor-addressable video display devices operating at a fairly high baud rate. CAVS currently does not support scrolling in any of its reports. Instead it displays one screenful of information, and then prompts the user to end the report, view the next screen of the report, or display a related report (and later return to the current screen). The user cannot back up and view the previous screen, but must end the report and start from the beginning. Vertical scrolling (on suitable high-speed video terminals) would be better. The user could simply scroll forward and backward as desired in the current screen. Horizontal scrolling would also be useful for a number of CAVS reports. For instance, the indentation of source code often forces long source lines to wrap around, reducing the number of source lines on the video screen. Note that CAVS could still generate interactive reports without scrolling on low-speed or hard-copy terminals.

4.2.2 Multiple Windows

The next step in enhanced user interfaces consists of multiple windows (report pages) on one video screen. User interfaces employing windows which open, expand, contract, scroll, and disappear are fast becoming accepted. Examples of systems which employ this concept are INTERLISP, SMALLTALK, the Apple LISA system, VisiON, and a number of other new microcomputer software products. The basic ideas seem to be:

1. Screen doesn't automatically scroll up because it never overfills.
2. One or more windows may be active at a time.
3. Windows are stacked and reappear at appropriate times.

4. User can move cursor around active window and point to parts of it.

5. Part or all of the data displayed in a window may be cutout and then pasted into something else.

6. Windows can be expanded or contracted by the user.

7. Windows can be scrolled horizontallyor vertically by the user.

8. A pointing device (mouse, light pen) replaces the need for a keypad and greatly reduces the number of keyboard entries required.

Scrolling and multiple windows are techniques which have been evolved for microcomputer systems (which usually have a high-speed video terminal and a low-speed printer). These techniques can be incorporated into CAVS for enhancing its user interface on selected video terminals, without degrading its user interface on other terminals.

4.2.3 Interactive CAVS Reports

In the approach currently used to provide interactive CAVS reports, the user directs what is to be displayed next, and the natural structure of the software is used to guide the user's choices. The structured walkthrough function of CAVS is currently the best example, but other reports could be handled in the same way. A significant advantage of this approach is that it does not presume a sophisticated video terminal - the structured walkthrough can be done from a hardcopy terminal. The availability of a video terminal with user-controllable scrolling and windows is then a plus, making it possible to view several interactive CAVS reports simultaneously.

4.3 EXPANDED WALKTHROUGH CAPABILITY

Use of CAVS with large project libraries has highlighted the following problems:

1. A project library can contain a wealth of information about a software system. Convenient access to this information, in the order that the user interactively selects, is a serious problem.

2. The project library will usually contain information about a large number of system-wide names. Not allowing the CAVS user to select a specific one of these names in a specific program-ID is a deficiency.

The following enhancements to make CAVS reports more interactive would help to solve these problems.

4.3.1 System Walkthrough

Calling tree reports for large systems quickly get out of hand, even for batch reports. An interactive approach would be to allow the user to walkthrough a calling tree, and to ask for more detailed information about that part of the tree on the screen. Information should be made available about the properties of formal parameters, their usage, and the text of the statements they occur in. The same kinds of information should also be available for global variables, and for files used in a program-ID. The information currently included in the CAVS module documentation report should also be available during an interactive system walkthrough. The user should be able to 'expand' a program-ID in the system walkthrough and view its internal perform-range structure as well as do a structured walkthrough within it.

4.3.2 Source Text Report Enhancements

CAVS displays selected source text for the interactive user, either during a structured walkthrough or as a specified statement in context (five preceding and five following statements). A number of enhancements to these (and related) modes of text display will increase the utility of CAVS for interactive users:

1. Horizontal scrolling to overcome the line wrap problem.
2. Vertical scrolling for more continuity in displaying source.
3. 'Expanding' symbols by name to give one of several possible reports.

4. String searching during the structured walkthrough for quick access to statements of interest.

The following 'new' source text reports are based on experience in using the current CAVS source text reports:

1. STRUCTURED WALKBACK - similar to structured walkthrough but in the opposite direction. This report would be especially useful for debugging and for increasing test coverage. While tracing backwards in the source code, the report would provide access to all information in the project library about global variables, files, and entry points. To enhance the usefulness of the structured walkback as a coverage assistance tool, testing history data for selected testcases could be displayed during the walkback. Note that this is an area where the ability to display several reports on the screen in multiple windows would make an outstanding contribution to the utility of CAVS.

2. PATH WALKTHROUGH/WALKBACK - similar to structured walkthrough/walkback, but well suited for unstructured code. This report would allow the interactive user to proceed through the code in a branch-by-branch analysis. CAVS would maintain a history of the branches taken to get to the current screen (allowing CAVS to backtrack for the user), as well as the branches not followed. Library-wide access to desired information and the ability to 'expand' PERFORM and CALL statements as in the structured walkthrough would be provided.

3. TRACE WALKTHROUGH/WALKBACK - similar to path walkthrough/walkback but with branch selection controlled by reading the trace file for a testcase. User controls for the amount of source to display would be in the form of breakpoints, or user selection of PROGRAM-IDs and perform ranges to display at the terminal. Note that these are necessary to prevent a deluge of output to the terminal and to control the speed of source text display.

4. SYMBOLIC WALKTHROUGH - an extension of path walkthroughs using symbolic execution to provide the current values for symbols upon user request at any point in the walkthrough. CAVS would essentially act as the bookkeeper for all symbols encountered along the current walkthrough. The user could then answer questions about the current value of individual variables without having to back up through the source text.

5. PSEUDOCODE WALKTHROUGH - pseudocode descriptions can be maintained on the same project library as the actual source. It could be printed in hardcopy reports or displayed in the structured walkthrough style. Note that this capability could also be used as a design tool. This report would really shine as a documentation and maintenance tool with a multiple window format because the user could walkthrough the real source code in one window and the psuedocode in another window (with other windows available for accessing library wide information at the same time).

6. CONTEXT REPORT - this is a combination of the structured walkthrough and cross-reference report. It would consist of displaying the context of the cross reference statements (the decision structure which leads up to the statements in the cross reference list). Scrolling with this display is also needed, as well as the ability to expand statements (in the sense of the structured walkthrough) which were left out.

4.4 INTEGRATE SOFTWARE LIFE CYCLE TOOLS IN THE AVS CONTEXT

Let's briefly forecast software life cycle technology in the AVS environment according to the following features:

1. FUNCTIONS - the list of AVS functions can be expanded to include editing, configuration control, management reporting, and incremental change analysis.

2. USER INTERFACE - in order to prevent the AVS menu system (and user's manual) from collapsing under its own weight, a set of software life cycle utilities is identified. Each operates from a

menu, and utilizes an extension of the CAVS project library as a central data base.

3. PERFORMANCE - emphasis is placed on maintaining current efficiency while increasing functions performed and ease of use.

4.4.1 Editing

CAVS is a very powerful tool for reading structured source code at a video terminal, but it does not include any way to modify the source as it is being read. This is a significant limitation of an otherwise powerful tool. We do not suggest incorporating an entire text-editor program into CAVS. Source code should still be created and large changes made with the user's favorite editor. But small changes for which it would be inconvenient to check-out a whole source module should be possible directly in CAVS. Also, some source-code editing functions may be found especially appropriate for CAVS (copying large blocks of existing text might be one).

CAVS Statement Editing Functions

1. Statement insertion/deletion - note that this is not the same as line insertion/deletion.
2. String search/replacement - the string search function would be useful during source text reading also.
3. Cutting/prefixing/appending/pasting - similar to word processing cut/paste buffers but more natural for working with structured blocks of text.
4. Pending statement(s) - to note that a particular section of code is not yet completed.
5. Cursor functions (video mode) - inserting/deleting characters at the cursor, deleting words.
6. Cursor movement - end of statement, beginning of statement, scroll up, scroll down, page forward/backward.
7. Copying - copying block of text from existing routines.

4.4.2 Configuration Control and Management Reporting

Configuration control and management is one of several directions in which to steer CAVS evolution. Configuration control consists of:

1. Multiple version identification
2. Change control
3. Status reporting

CAVS emphasizes the importance of a project library which contains source text, as well as interface information and testing history information, so that meaningful interactive reports can be generated. Storing the same source text outside CAVS is redundant and expensive, especially in large software systems that exist in several versions. CAVS can ease the configuration problem and save money for its users if it is made able to support multiple versions. Note that this will provide not only multiple versions of source text but also multiple versions of all displays, reports, and testing histories.

The following is a suggested list of configuration control functions:

1. DIFFERENCES to automatically find the lines that are different in the old and new versions of a COBOL routine. COBOL source text differences (by section and by perform range) will be more concise than those produced by general-purpose difference routines.
2. CHECK OUT/IN to coordinate changes to the same COBOL routine. Allow the project manager to decide how many copies of the same routine can be checked out at the same time. Specific job control language may be put before and after modules as they are checked out.
3. VERSIONS to produce complete source text listings as well as all project library reports for ALL versions. Given an earlier and derived version name, CAVS can produce the the differences between the two (in terms of source line differences as well as project library differences).

4. PROBLEM REPORTS for multiple versions can be maintained on the project library.

Other management functions, to be provided by the host environment if possible are:

- Management reports.
- Interfaces with other tools.
- Limitation of read/write privileges to specific areas of a project library.
- Support of concurrent updates to the same library.
- Merging several new versions of the same routine.
- Storing object(relocatables), execute(absolutes), job control language, test data, and other user-defined data which is not strictly source code.

Menus to Support Multiple Versions

The Functions menu in CAVS could be modified to have VERSION SELECTION and VERSION DIFFERENCES submenus. In the following example they are items 14 and 15.

```
--FORTRAN FUNCTIONS--------LIBRARY FUNCTIONS----------OTHER FUNCTIONS---
1 - READ SOURCE          7 - STRUCTURED WALKTHROUGH 13- DEFINE REPORT FORMAT
2 - ERRORS/WARNINGS      8 - SHOW LIBRARY CONTENTS  14- VERSION SELECTION
3 - PRECOMPILE/INSTRMT   9 - SYSTEM REPORTS         15- VERSION DIFFERENCES
4 - MODULE REPORTS      10 - COVERAGE ANALYSIS      16- END OF CAVS SESSION
5 - SYMBOL REPORTS      11 - TRACE FILE REPORTS
6 - REFORMAT            12 - CHECK OUT SOURCE
----------------------------------------------------------------------
NEW VERSION = (VAX        )          OLD VERSION = (BASE LINE      )
----------------------------------------------------------------------
```

The "new version" indicated at the bottom of the menu indicates the version name to be associated with any new or modified source read in during the current session. The submenus might appear as follows:

```
-------------VERSION SELECTION------------CURRENTLY-------------
0 - RETURN TO FUNCTION MENU
1 - SELECT OLD VERSION (WILDCARD PROMPT)  BASELINE
2 - SPECIFY A NEW VERSION                 (NONE SPECIFIED)
3 - VERSION OVERVIEW

-----------VERSION DIFFERENCES------------CURRENTLY-------------
0 - RETURN TO FUNCTION MENU
1 - SELECT VERSION A                        VAX
2 - SELECT VERSION B                        IBM
3 - TEXT DIFFERENCES
4 - CALL DIFFERENCES
5 - COMMON BLOCK DIFFERENCES
6 - COMDECK DIFFERENCES
7 - FILE DIFFERENCES
8 - TESTCASE DIFFERENCES
```

4.4.3 Incremental Change Analysis

Incremental change analysis means not having to reanalyze each compilation unit that has changed, but only each modified statement.

1. Incorporate text editing features into CAVS.

2. Analyze changed lines only, rather than entire PROGRAM-IDs which have changed.

5 SPECIAL COMMENTS

This section summarizes the knowledge gained during the CAVS development (as well as earlier AVS developments) sponsored by RADC and DMA. It is intended to assist designers and implementers of AVS tools, as well as anyone modifying or enhancing CAVS (or any of the other AVS tools discussed). The table below summarizes the history and overall features of the AVS tools developed:

OVERVIEW	JAVS	FAVS.0	FAVS.4	CAVS
When was the AVS completed ?	1975	1978	1980	1983
What was the approximate cost?	$200k	$120k	$ 70k	$500k
Number of initial installations?	1	3	2	5
Number of current installations?	4	1-15	3-10	5
Language processed?	JOVIAL J3	FORTRAN	FORTRAN	COBOL
Number of dialects processed?	1	2	2	6
Structured extensions processed?	none	DMATRAN	DMATRAN	PSL
Data base extensions processed?	none	none	none	SYS-2000 DMS-1100
Batch/interactive/both	batch	batch	batch	both
Written in	JOVIAL J3	DMATRAN	DMATRAN	DMATRAN
Approximate number source lines	25k	35k	35k	45k

Real-world use of FAVS.0 and FAVS.4 has shown that AVS tools such as these are valuable aids in software development. The major hindrance to more widespread use of these tools has been lack of coordinated and continuous maintenance for the tools. This situation should improve for FAVS.4 and CAVS.

The Defense Mapping Agency (DMA) is currently maintaining FAVS.4, plans to maintain CAVS, and plans to continue the maintenance for several years.

Maintenance directly relates to reliability, the most important feature for these tools to have. The next most important features are what functions they perform and how efficiently they perform them. When AVS tools are used in large software development projects, it is essential that the tools maintain a data base describing the total software system. This data base will naturally be large and complex. Its contents are crucial in determining the functions that can be performed and what it costs to perform them. The next table briefly summarizes the data base contents of each AVS tool:

DATA BASE CONTENTS	JAVS	FAVS.0	FAVS.4	CAVS
Detailed module tables	yes	yes	no	no
Source text	yes	yes	no	yes
Testing history	no	no	no	yes
Copy Text source	NA	no	no	yes
Global name cross-reference	no	no	yes	yes

The chart above refers to the contents of the data base permanently maintained by the tools. All of the tools construct detailed module tables for the source code that is read in; JAVS and FAVS.0 save them on the run-to-run data base. FAVS.4 introduced the interface library concept, extracting the global and interface information from the detailed module tables and saving that rather than the much larger detailed information. A simple analogy may help to illustrate this concept. Reading large amounts of source text and extracting global and interface information is analogous to separating wheat from chaff. You end up with a big pile of chaff and a small pile of wheat. Before the interface library concept, the AVS tools would mix the two piles together at the end of a run. The interface library concept is simply to save the wheat and throw the chaff away. The chart below indicates the major impact that this approach has on the cost of using the tool on large software developments.

RESOURCE USAGE & SIZE CONSTRAINTS	JAVS	FAVS.0	FAVS.4	CAVS
AVS time/compilation time	10-100	5-100	1.5-3	1-2
Library size/source size	5	10	.3	1.5
Maximum modules per library	250	250	no limit	no limit
Maximum statements per library	5k-10k	5k-10k	500k	100k-200k

The tools which maintain a large and detailed data base become exponentially more expensive to run on larger amounts of software. The mean time between failures on the host computer becomes the determining factor (as well as the available computer budget) in how large a library can be built. By introducing the interface library concept, FAVS.4 greatly increased its usability in exactly those situations where it is of most value - large software developments. CAVS extended the interface library concept to that of the project library which contains all global and interface information as well as the source code (but not detailed module tables) and testing history. Retaining the source text in CAVS project libraries is essential because CAVS operates in interactive mode. Interactive users want to see their source code, not the statment numbers that refer to the source code.

The next most important feature of the AVS tools is their user interface: how easy are they to use? The chart below summarizes the batch user interfaces of the tools.

BATCH COMMAND PROPERTIES	JAVS	FAVS.0	FAVS.4	CAVS
Report oriented/process oriented	process	report	report	report
Report index generated by tool?	no	yes	yes	yes

The essential lesson learned here is that batch users should tell the tool what they want (in terms of reports), not the process or sequence of steps required to produce the reports.

CAVS is the only one of the four AVS tools that supports interactive use. CAVS provides batch commands for batch usage, and menus/help screens for interactive use. CAVS menu commands are also available for expert CAVS users. The chart below summarizes the interactive interface properties.

INTERACTIVE INTERFACE PROPERTIES	JAVS	FAVS.0	FAVS.4	CAVS
Menus/commands	none	none	none	both
Number of menus	0	0	0	35
Number of help screens	none	none	none	200
Name selection(full name/wildcard)	full	full	full	both

The functions performed by the AVS tools are best indicated by the reports which they produce and the analysis they support. Nine types of reports, features, and analysis are distinguished:

	REPORT/FEATURE/ANALYSIS	DESCRIPTION
1.	Library contents reports	Reports which describe the permanent data base
2.	Source listing features	Additional features of source code listings
3.	Detailed module reports	Reports about individual compilation units
4.	System documentation	Reports about the whole software system
5.	Error/warning functions	Additional static analysis features
6.	Precompilation functions	Source code transformations
7.	Dynamic analysis functions	Analysis of run-time behavior
8.	Reformatting functions	Readability/maintainability transfor-mations
9.	Selective report generation	Reports about selected individual names

The table below summarizes the total number of reports/features/functions for each AVS tool. The growth in the number of functions supported is evident by comparing JAVS (the first AVS tool) to CAVS (the most current tool). JAVS had 22 total reports/features/functions, while CAVS has 75. Each of the nine categories is elaborated in the charts which follow.

REPORTS/FEATURES/FUNCTIONS	JAVS	FAVS.0	FAVS.4	CAVS
Total library reports	2	2	2	8
Additional listing features	1	1	2	9
Total detailed reports	4	6	7	13
Total system reports	4	7	8	17
Total error/warning features	0	4	4	7
Total precompilation functions	4	1	1	6
Total dynamic analysis functions	4	2	2	6
Total reformatting functions	1	1	1	1
Total name selections	2	1	1	8
Total reports/features/functions	22	25	28	75

LIBRARY CONTENTS REPORTS	JAVS	FAVS.0	FAVS.4	CAVS
Modules/entry points	yes	yes	yes	yes
Commons/compools	no	no	no	yes
Copy text/includes	NA	no	no	yes
Entry points defined/not called	no	no	no	yes
Entry points called/not defined	no	no	no	yes
Testcases	no	no	no	yes
Files	no	no	no	yes
Current modules	yes	yes	yes	yes
Total library reports	2	2	2	8

SOURCE LISTING FUNCTIONS	JAVS	FAVS.0	FAVS.4	CAVS
Indent source	yes	yes	yes	yes
Reconstructed/Original source	recon	recon	orig	orig
Internal procedure XREF	no	no	no	yes
List a range of statements	no	no	no	yes
List a depth of statements	no	no	no	yes
Expand internal procedure calls	no	no	no	yes
Expand external procedure calls	no	no	no	yes
Omit/expand nested code	no	no	no	yes
Access to other reports	no	no	no	yes
Additional listing features	1	1	2	9

DETAILED DOCUMENTATION REPORTS	JAVS	FAVS.0	FAVS.4	CAVS
Picture report	yes	no	yes	yes
Statement profile report	no	yes	yes	yes
Reaching set report	yes	yes	yes	yes
Symbol properties report	no	yes	yes	yes
Symbol Xref report	yes	yes	yes	yes
Symbol source report	no	no	no	yes
Labels Xref report	no	no	no	yes
Labels source report	no	no	no	yes
Constants Xref report	no	no	no	yes
Constants Source report	no	no	no	yes
Externals Xref report	no	yes	yes	yes
Externals source report	yes	yes	yes	yes
Module Documentation report	no	no	no	yes
Total detailed reports	4	6	7	13

SYSTEM DOCUMENTATION REPORTS	JAVS	FAVS.0	FAVS.4	CAVS
Symbol Xref	yes	no	yes	yes
Common/Compool Summary	no	yes	yes	yes
Common/Compool Matrix	no	yes	yes	yes
Common/Compool Xref	no	yes	yes	yes
Common/Compool Source	no	no	no	yes
Copy Text Summary	NA	no	no	yes
Copy Text Matrix	NA	no	no	yes
Copy Text Xref	NA	no	no	yes
Copy Text Source	NA	no	no	yes
File Summary	no	no	no	yes
File Matrix	no	no	no	yes
File Xref	no	no	no	yes
File Source	no	yes	yes	yes
Entry Point Summary	yes	yes	yes	yes
Entry Point Bands	yes	yes	yes	yes
Entry Point Tree	yes	no	no	no
Entry Point Xref	no	yes	yes	yes
Entry Point Source	no	no	no	yes
Total system reports	4	7	8	17

ERROR/WARNING FUNCTIONS	JAVS	FAVS.0	FAVS.4	CAVS
Call Checking	no	yes	yes	yes
Graph Checking	no	yes	yes	yes
Mode/Picture Checking	no	yes	yes	yes
Set/Use Checking	no	yes	yes	yes
Data Flow Checking	no	no	no	yes
Access to source of called routines	no	no	no	yes
Access to symbol reports	no	no	no	yes
Total error/warning features	0	4	4	7

PRECOMPILATION FUNCTIONS	JAVS	FAVS.0	FAVS.4	CAVS
Structured Dialect Precompilation	no	no	no	yes
Control Flow Instrumentation	yes	yes	yes	yes
Assertion Instrumentation	yes	no	no	no
Trace Instrumentation	yes	no	no	yes
Timing Instrumentation	yes	no	no	yes
File I/O Instrumentation	no	no	no	yes
Check-out source code	no	no	no	yes
Total precompilation functions	4	1	1	6

DYNAMIC ANALYSIS FUNCTIONS	JAVS	FAVS.0	FAVS.4	CAVS
Control flow summary	yes	yes	yes	yes
Control flow counts	yes	yes	yes	no
Control flow source	yes	no	no	yes
Control flow history	no	no	no	yes
Abort trace	no	no	no	yes
Timing	yes	no	no	yes
File I/O	no	no	no	yes
Total dynamic analysis functions	4	2	2	6

REFORMATTING FUNCTIONS	JAVS	FAVS.0	FAVS.4	CAVS
Reformat source code	yes	no	no	yes
Restructure source code	no	yes	yes	no
Total reformatting functions	1	1	1	1

SELECTIVE REPORT GENERATION	JAVS	FAVS.0	FAVS.4	CAVS
By Module name	yes	yes	yes	yes
By Entry Point name	yes	no	no	yes
By Symbol name	no	no	no	yes
By Label name	no	no	no	yes
By Constant name	no	no	no	yes
By Common/Compool name	no	no	no	yes
By Copy text name	no	no	no	yes
By File name	no	no	no	yes
By Testcase name	no	no	no	yes
Total name selections	2	1	1	8

E

ED

8